Tic-Tac-Toe by Reinforcement

Nathaniel Beckemeyer

1 Introduction

This project used reinforcment learning to play Tic-Tac-Toe, and specifically the Sarsa and Q-Learning algorithms, and their respective λ versions. I used Sam Beckmann's modified framework for this project, rather than the one distributed to us.

In this report, the probability of an intermediate reward being given after an agent makes a move is broken down into four psssibilities according to Table 1. Note that intermediate rewards are always provided at the end of games.

	Probability	
Agent Type	(a)	(b)
Deterministic	1	0
Nondeterministic	0.75	0.50

Tab. 1: Description of behaviors

The section that follows is highlights and a brief summary of the results of the tests that were run and a short discussion. Appendix A contains more detailed figures of the runs of the RL players against the optimal players. See Appendix B for the reinforcement learning agents playing themselves.

2 Results & Discussion

The results visualized in this report use a 'win percentage' metric; this percentage is a periodic average; every 50 games the summary data for the percentages were calculated.

Overall Q-Learning and Sarsa perform similarly—I could notice no substantial differences in their performances.

The Q-Learning agent learning to defeat the naive agent (in this case, a random agent) can be seen in Figure 1. Similarly, the performance of the Sarsa agent vs the naive agent can be seen in Figure 2.

The most startling result was that both of the λ players were outperformed by their non-λ counterparts! I'm not sure exactly why this happened, but fortunately the λ versions do learn to tie the optimal player. Except for one; interestingly, the Sarsa (λ) player actually loses to the non-deterministic minimax

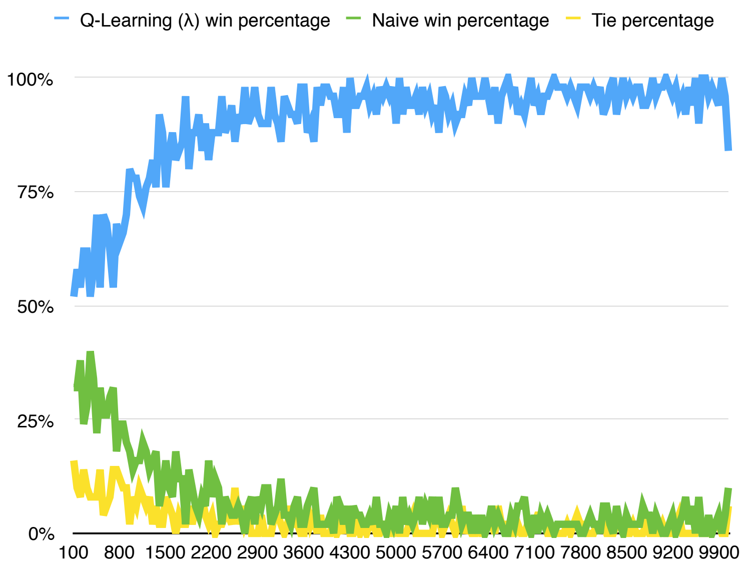

Fig. 1: QLearning vs the naive player

player with intermediate rewards. I suspect that the reason for this loss is that Sarsa does not deal well with the intermediate rewards (which can be incorrect, as they are based on a minimax and do not distinguish between multiple best moves), hence why Sarsa generally performs better in the (b) conditions.

Doing this task taught me about the subtle difference and immense similarity between Sarsa and Q-learning. It also taught me just how quickly RL agents can learn in small state spaces—the speed difference between my agents and minimax made me wonder at what point it makes sense to use an RL learner rather than just a hardcoded solution.

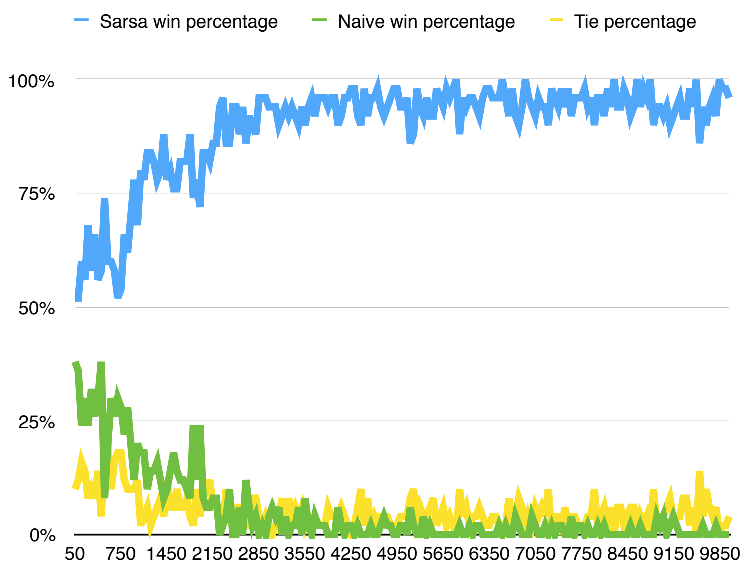

Fig. 2: Sarsa vs the naive player

A Optimal player results

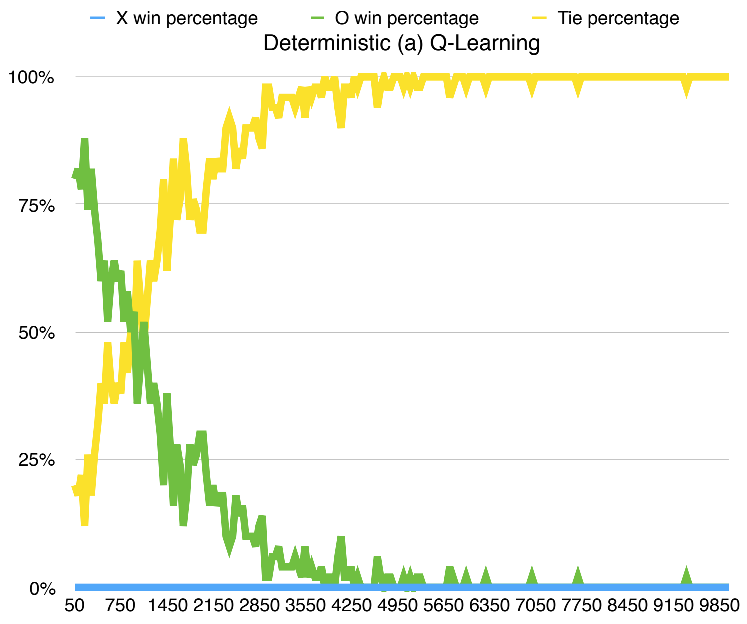

Fig. 3: QLearning with deterministic behavior (a)

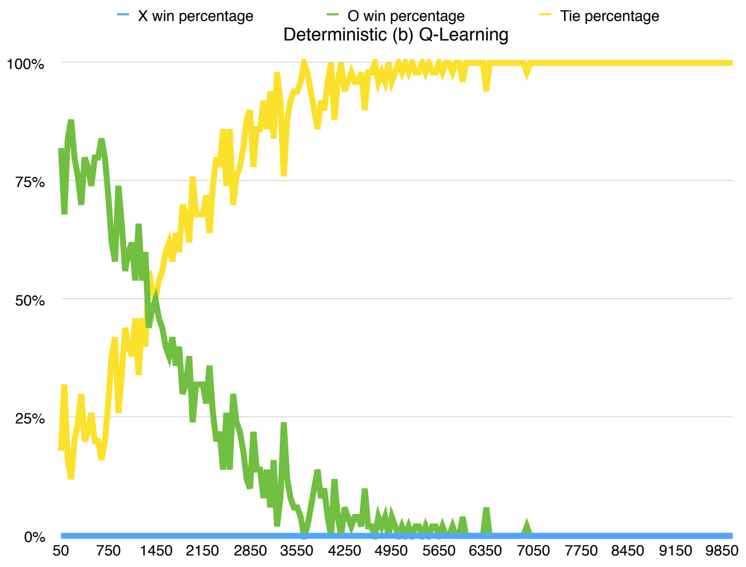

Fig. 4: QLearning with deterministic behavior (b)

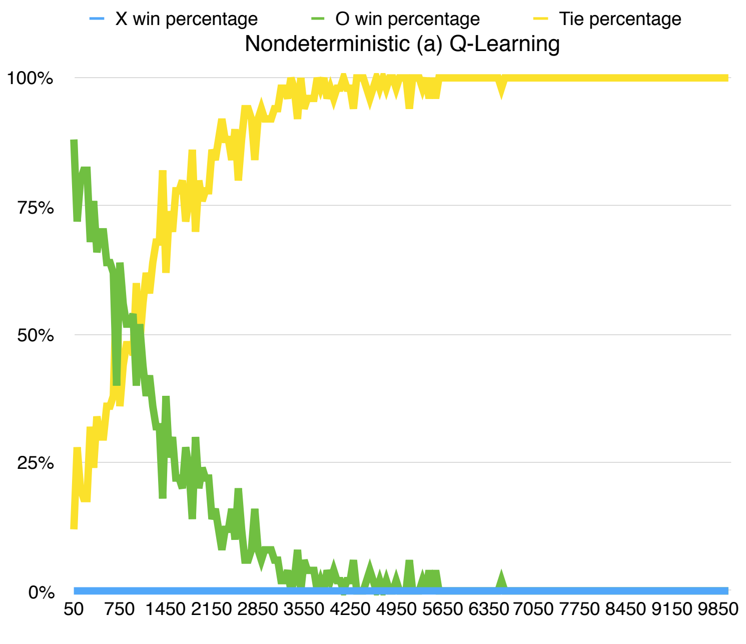

Fig. 5: QLearning with nondeterministic behavior (a)

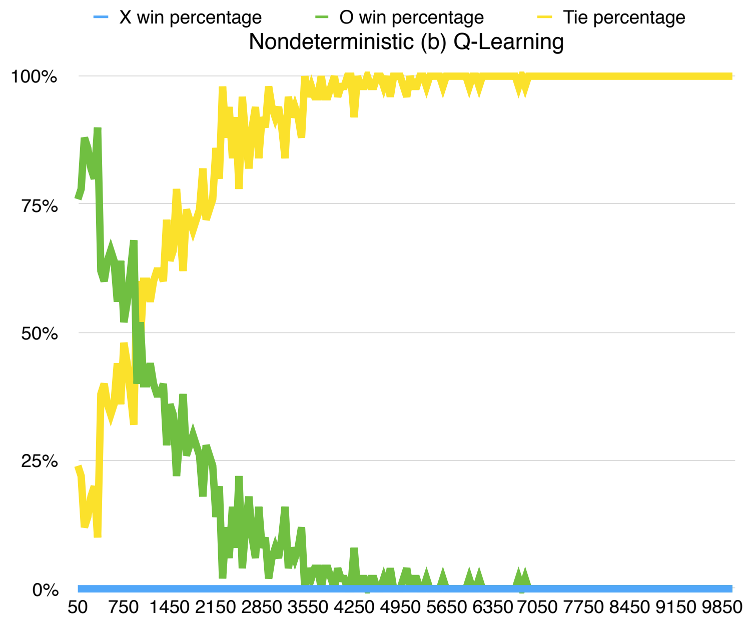

Fig. 6: QLearning with nondeterministic behavior (b)

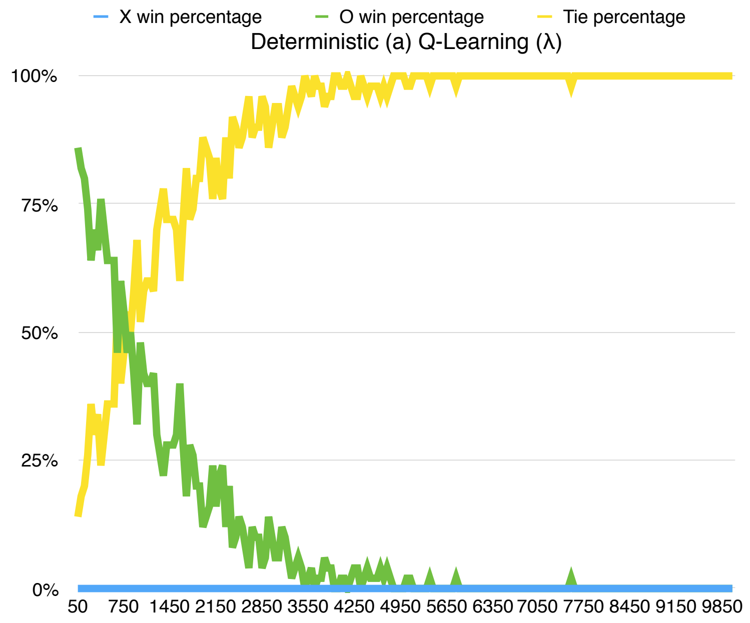

Fig. 7: QLearning (λ) with deterministic behavior (a)

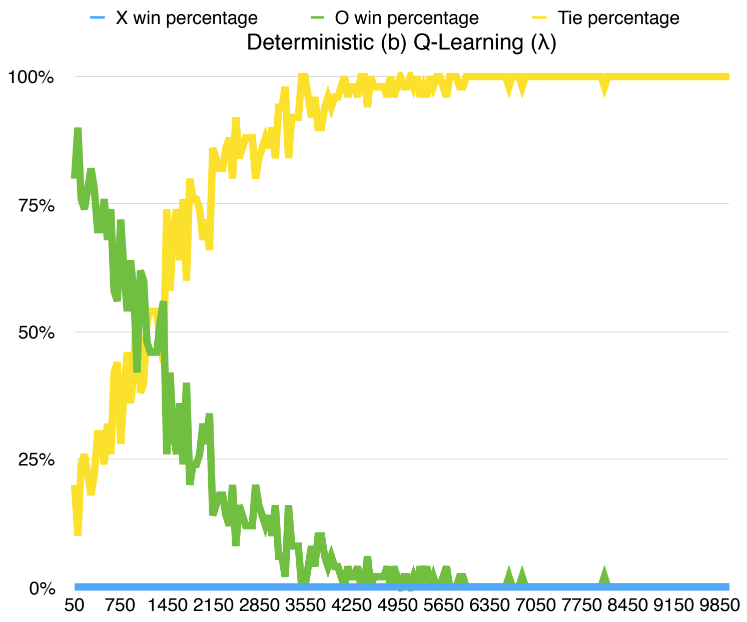

Fig. 8: QLearning (λ) with deterministic behavior (b)

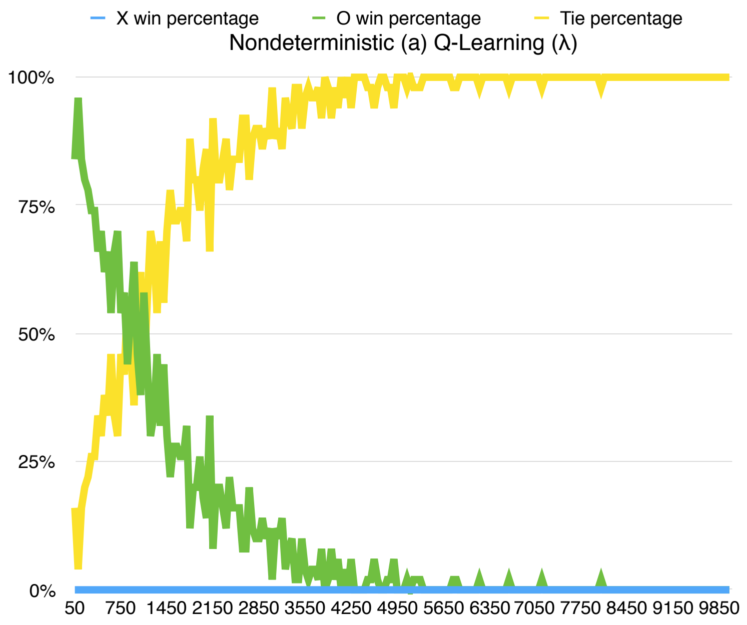

Fig. 9: QLearning (λ) with nondeterministic behavior (a)

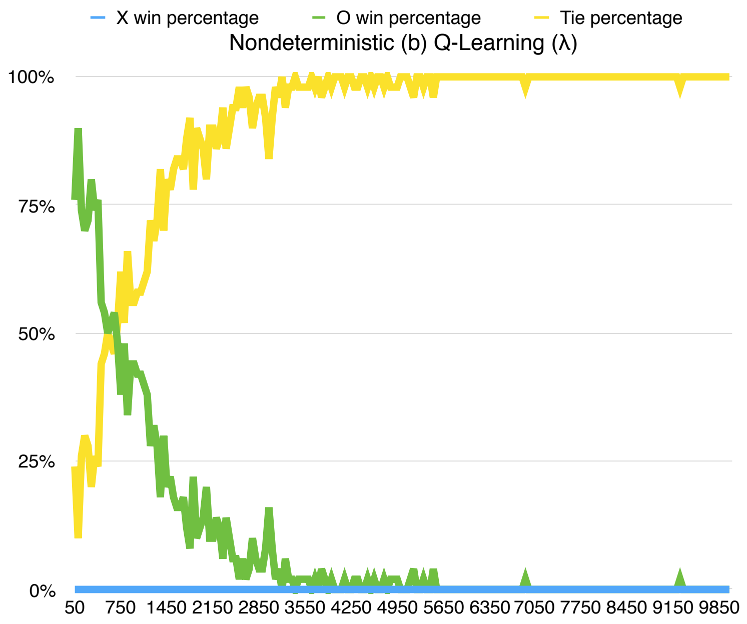

Fig. 10: QLearning (λ) with nondeterministic behavior (b)

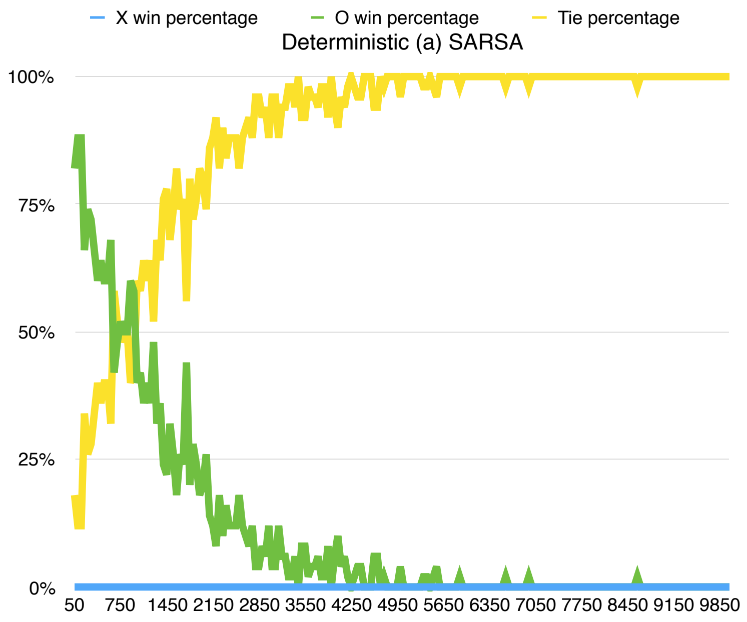

Fig. 11: Sarsa with deterministic behavior (a)

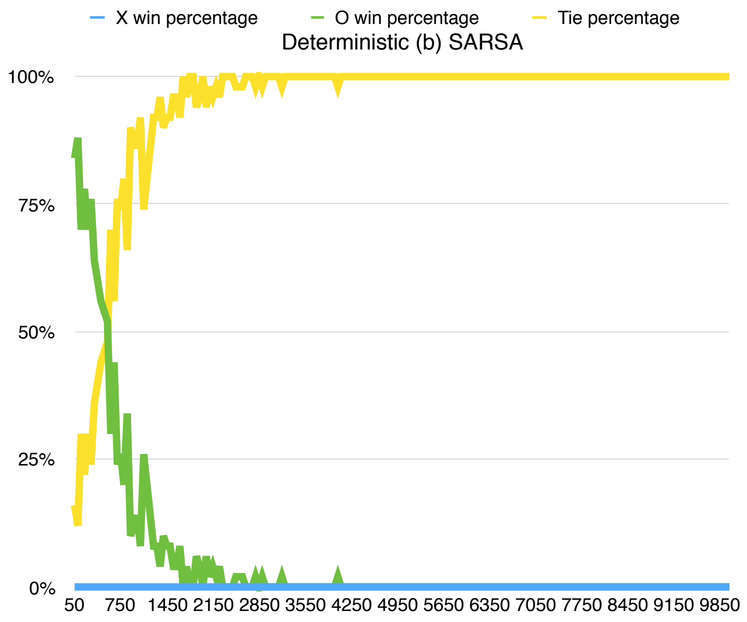

Fig. 12: Sarsa with deterministic behavior (b)

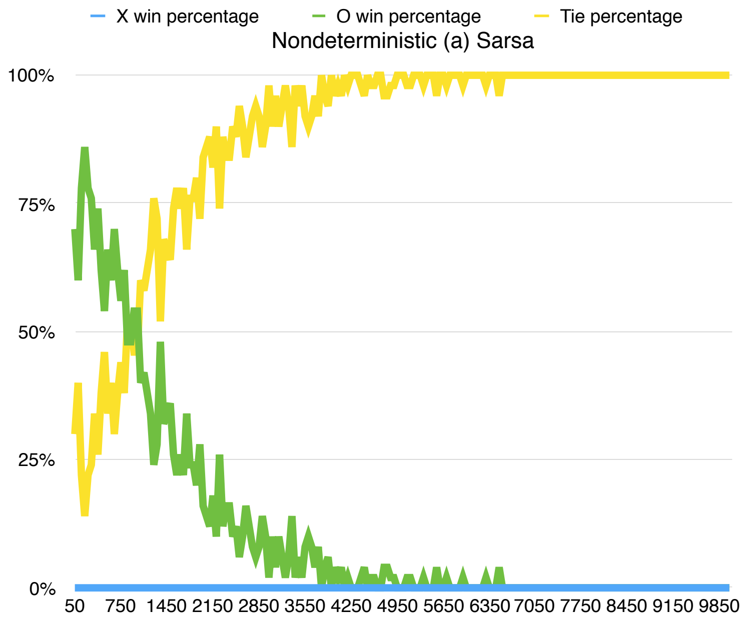

Fig. 13: Sarsa with nondeterministic behavior (a)

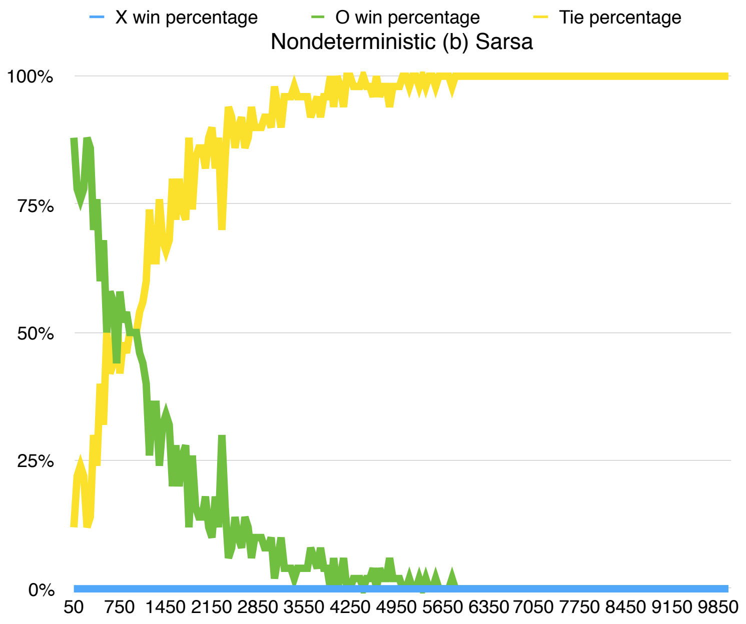

Fig. 14: Sarsa with nondeterministic behavior (b)

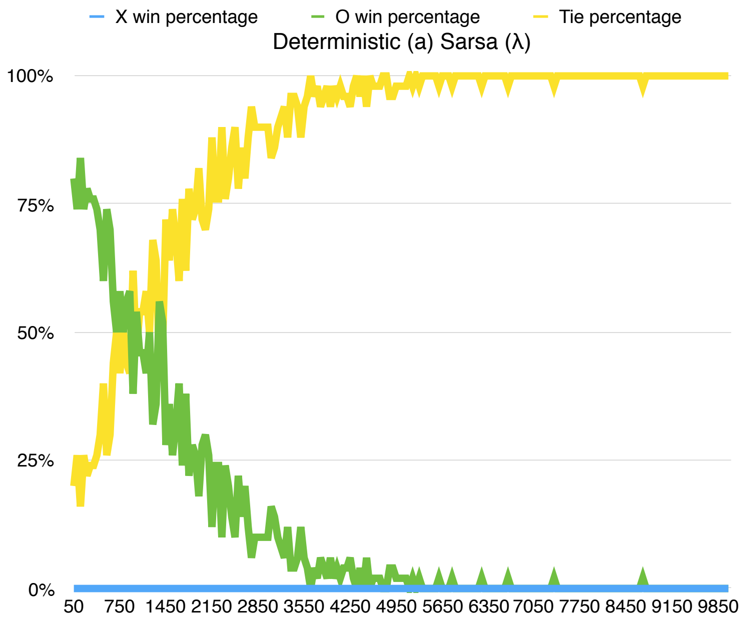

Fig. 15: Sarsa (λ) with deterministic behavior (a)

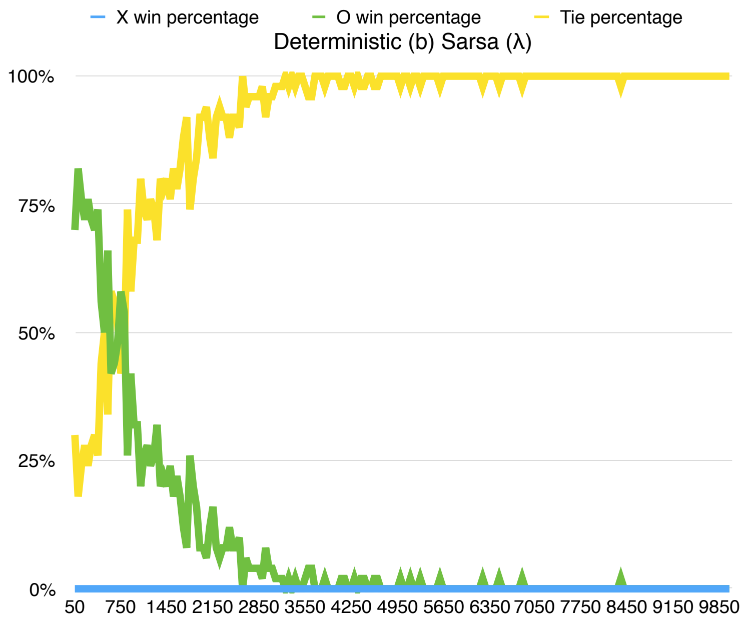

Fig. 16: Sarsa (λ) with deterministic behavior (b)

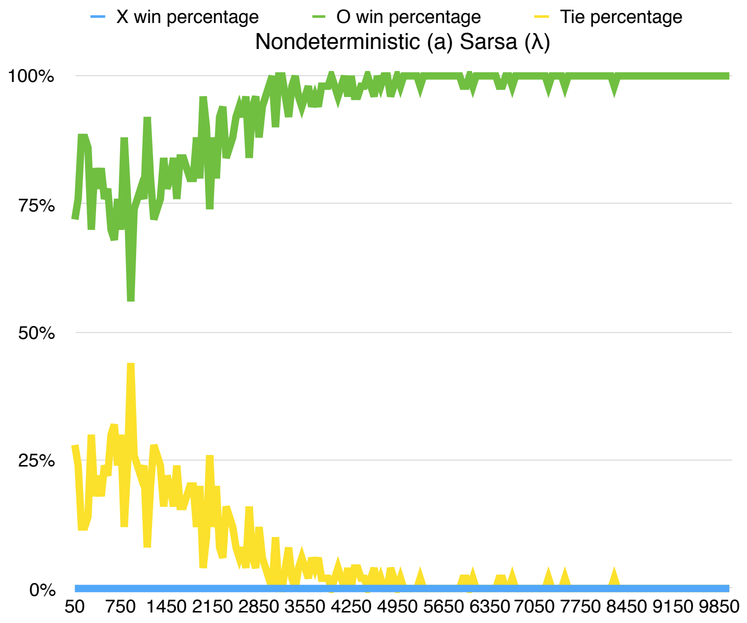

Fig. 17: Sarsa (λ) with nondeterministic behavior (a)

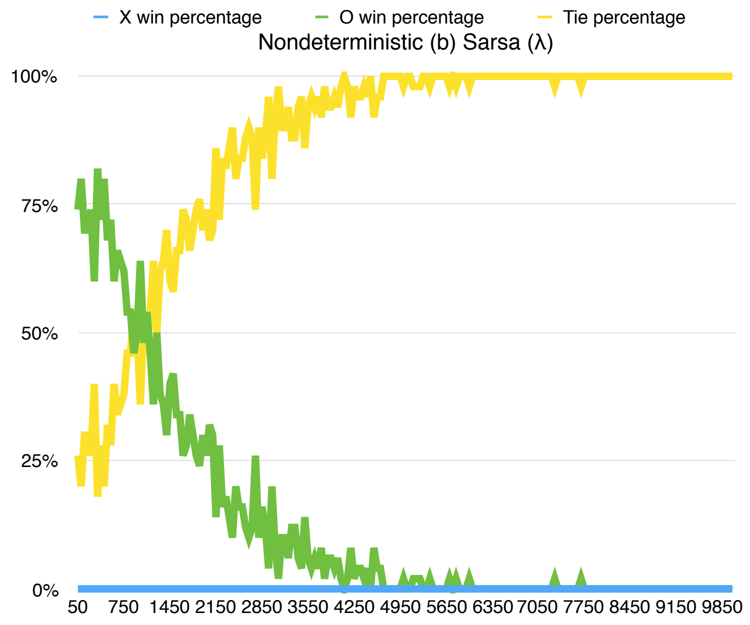

Fig. 18: Sarsa (λ) with nondeterministic behavior (b)

B RL against itself

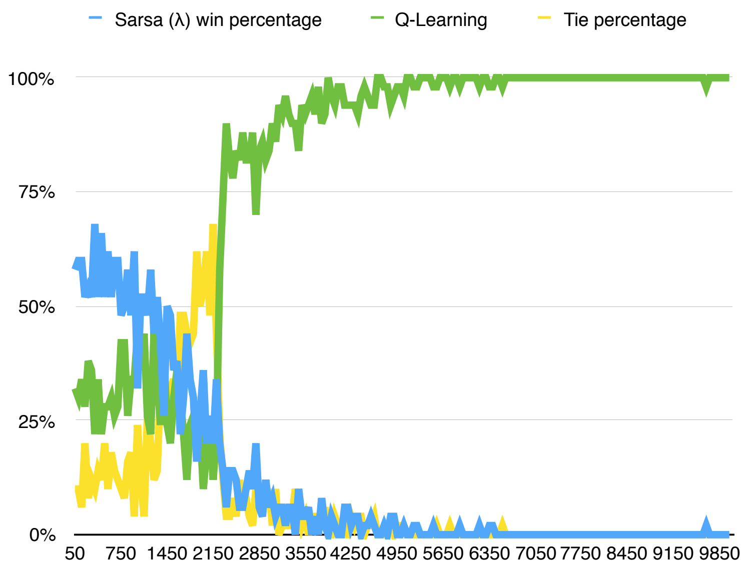

Fig. 19: Sarsa (λ) against a Q-Learner

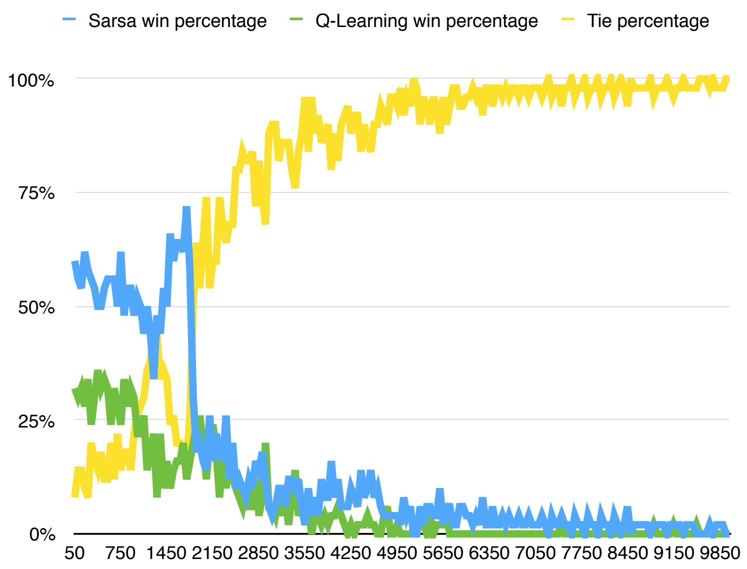

Fig. 20: Sarsa against a Q-Learner

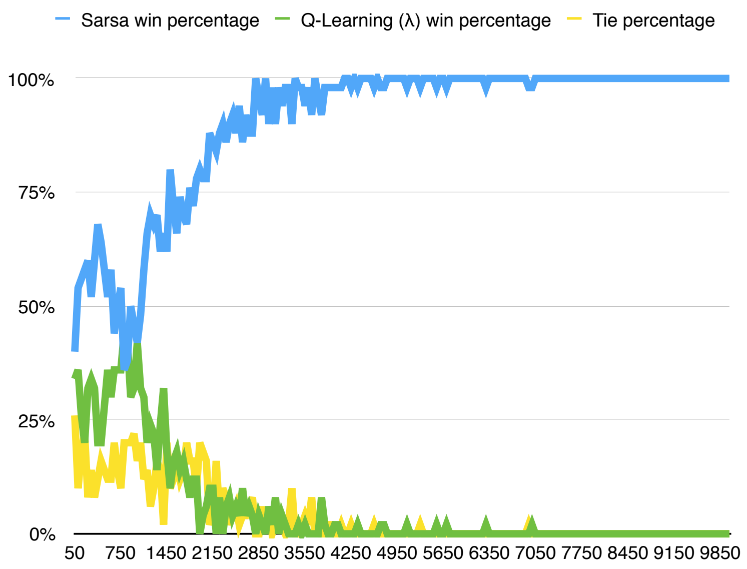

Fig. 21: Sarsa against a Q (λ)-Learner

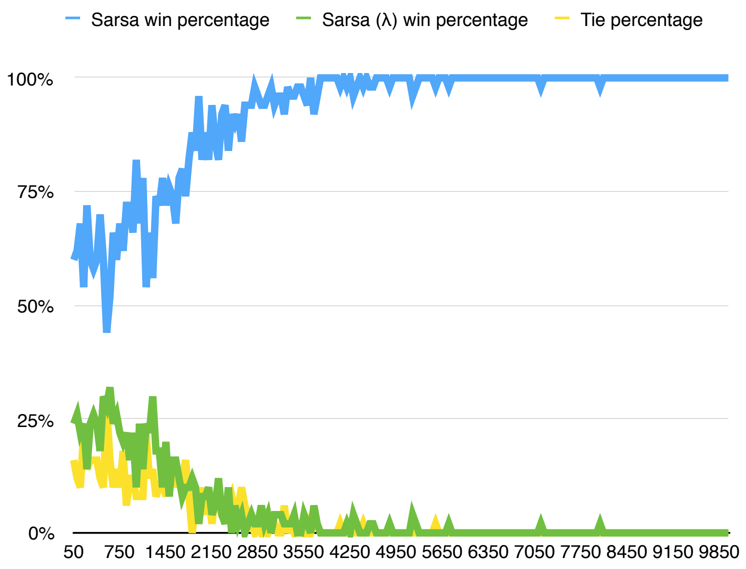

Fig. 22: Sarsa against a Sarsa (λ)-Learner

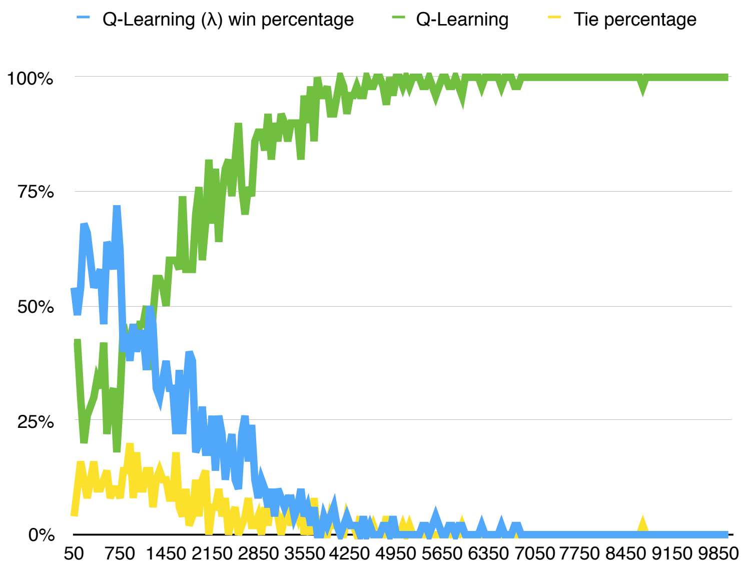

Fig. 23: Q(λ) against a Q-Learner

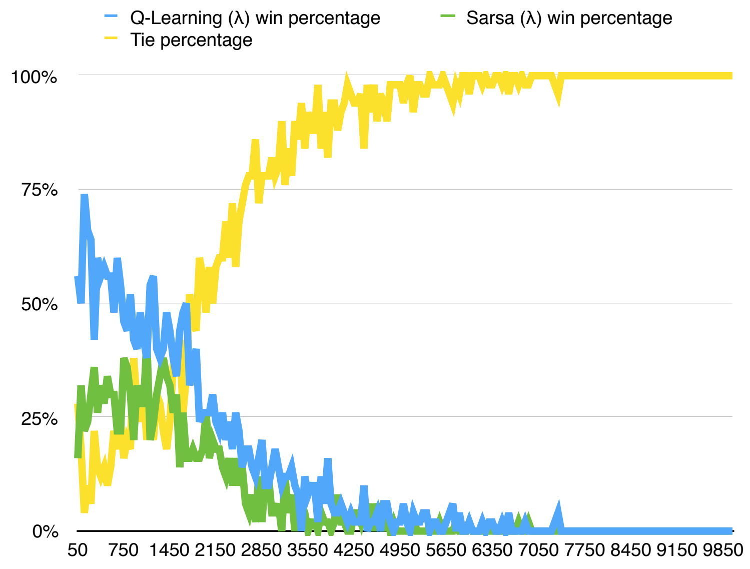

Fig. 24: Q(λ) against a Sarsa (λ)-Learner